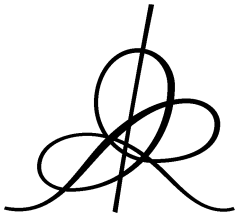
METHOD AND SOFTWARE FOR FAST CONSTRUCTION OF PRINCIPAL MANIFOLDS APPROXIMATIONS

Andrei ZINOVYEV



Institut des Hautes Études Scientifiques
35, route de Chartres
91440 – Bures-sur-Yvette (France)

Octobre 2003

IHES/M/03/61

METHOD AND SOFTWARE FOR FAST CONSTRUCTION OF PRINCIPAL MANIFOLDS APPROXIMATIONS

Andrey Zinovyev
Institut des Hautes Études Scientifiques (IHES), Bures-sur-Yvette, France
zinovyev@ihes.fr

Abstract

We propose a new algorithm for fast construction of grid approximations of principal manifolds with given topology. One advantage of the method is a new form of the functional to be minimized, which becomes quadratic at the step of refinement of the vertexes positions. This allows very effective implementations of the algorithm; the computational complexity grows linearly with the number of points with relatively small constant factor. Another advantage is that the same algorithmic kernel is applied to construct principal manifolds of different dimensions and topologies. We demonstrate how flexibility of the approach allows easily numerous adaptive strategies like principal graph constructing, etc. The algorithm is implemented as a C++ package. We describe the approach and provide several examples with speed performance characteristics.

Introduction

Principal manifolds were introduced by Hastie and Stueltze in 1989 as lines or surfaces passing through "the middle" of the data distribution [Hastie and Stuetzle, 1989]. This intuitive notion, corresponding to the human brain generalization ability, was supported by a mathematical notion of self-consistency: every point of the principal manifold is a conditional mean of all points that are projected into this point. Since in case of datasets only one or zero data points are projected in a typical point of the principal manifold, one has to introduce smoothers that become an essential part of the principal manifolds construction algorithms.

Since the pioneer work of Hastie, many modifications and alternative definitions of the principal manifolds appeared in the literature. Theoretically, existence of self-consistent principal manifold is not guaranteed for any probability distribution, and because of this many alternative definitions were introduced [Kegl, 1999], allowing constructing principal curves (manifolds), given that the distribution of points has several finite first moments. One of the most computationally effective and robust algorithmic kernel for principal curves construction, called Polygonal algorithm, was proposed in [Kegl et al., 1999]. Also a variant of strategy for constructing principal graphs was formulated in [Kegl et Krzyzak, 2002] in the context of the skeletonization of hand-written digits. Another interesting approach we would like to mention is constructing principal manifold in a piece-wise manner by fitting unconnected line segments [Verbeek et al., 2000].

Probably, the most of scientific and industrial applications of principal manifolds ideology were implemented using the SOM approach, coming from the theory of neural networks [Kohonen, 1982]. These applications are too numerous to be mentioned here. We only mention that the SOM strategy, indeed, can provide principal manifolds approximations (for example, see [Ritter et al.,1992; Mulier and Cherkassky, 1995]) and is computationally effective. The disadvantage of the approach is that it is entirely based on heuristics; also it was shown that in the SOM strategy there does not exist any objective function that is minimized by the training process [Erwin et al., 1992].
Our goal in this paper is to introduce a computationally effective way of principal manifolds construction. Our approach is similar to the work of Kegl, taking some features from the SOM approach as well. We use grid approximations to the principal manifold. The topology of the manifold can be fixed, or one can use principal graphs construction strategy, similar to the methods developed in [Kegl, 1999].

One difference of our approach from that one of Kegl is introducing two new smoothness penalty terms, which are quadratic at the vertexes optimization step. This allows using standard minimization of quadratic functionals (i.e., solving a system of linear algebraic equations with a sparse matrix), which is considerably more computationally effective than gradient optimization of more complicated function, introduced by Kegl. Moreover, quadratic functionals can be minimized by every space coordinate independently, giving possibility to use parallel minimization that is expected to be particularly effective in case of multidimensional data.

Other feature of our approach is a universal and flexible way to describe the grid. The grid approximation to the principal manifold is defined as a connected graph of nodes placed in data space and having some "natural" nodes placement in a low-dimensional space. Then the same algorithmic kernel is used to optimize this graph with respect to the dataset. Thus, the same algorithm, given an initial definition of the grid, provides construction of principal manifolds with different dimensions and topologies.

Our algorithm is implemented as a C++ package ***elmap*** available at *http://www.ihes.fr/~zinovyev/vidaexpert/elmap* and as a stand-alone application ***VidaExpert*** for multidimensional data visualization, available at *http://www.ihes.fr/~zinovyev/vidaexpert/vidaexpert.htm*. Applications of the approach to the data visualization were described in [Gorban and Zinovyev, 2001; Gorban et al., 2001, 2003].

General schema of the method

Lets define *elastic net* as a connected unordered graph $G(Y,\boldsymbol{E})$, where $Y = \{y^{(i)}, i=1..p\}$ denotes collection of graph nodes, and $\boldsymbol{E}=\{E^{(i)}, i=1..s\}$ is the collection of graph edges. Let's combine some of the adjacent edges in pairs $R^{(i)} = \{E^{(i)},E^{(k)}\}$ and denote by $\boldsymbol{R}=\{R^{(i)}, i=1..r\}$ the collection of *elementary ribs*.

Every edge $E^{(i)}$ has the beginning node $E^{(i)}(0)$ and the ending node $E^{(i)}(1)$. Elementary rib is a pair of adjacent edges. It has beginning node $R^{(i)}(1)$, ending node $R^{(i)}(2)$ and the central node $R^{(i)}(0)$ (see Fig. 1).

Introducing edges is simply introducing connectivity on the graph; this connectivity defines a topology of principal manifold to be constructed, as well as its dimensionality. Ribs together with edges are used to define smoothness penalty function, in such a way defining a "natural" form of the grid.

Figure 2 illustrates some examples of the grids practically used. The first is a simple polyline, the second is planar rectangular grid, third is planar hexagonal grid, forth – non-planar graph with nodes arranged on the sphere (spherical grid), then a non-planar cubical grid, torus and hemisphere. Elementary ribs at these graphs are adjacent edges intersecting with a blunt angle.

We underline here that the grids presented on fig.2 have different internal dimensionality and topology. This topology is optimized in data space with respect to the data point positions.

Formulating optimization criterium, we use widely used mean squared point-to-node distance as a main term, and two penalty terms, which are useful to interpret in terms of physical elastic properties of the grid.

On the graph G we define the energy function U that summarizes energies of every node, edge and rib:

$$U = U^{(Y)} + U^{(E)} + U^{(R)}. \quad (1)$$

Let's divide the data points into subcollections $K^{(i)}$, $i = 1\ldots p$, each of them contains data points for which the node $y^{(i)}$ is the closest one:

$$K_i = \{x^{(j)} : \left\|x^{(j)} - y^{(i)}\right\| \le \left\|x^{(j)} - y^{(m)}\right\|, m = 1,\ldots, p\}.$$

Also let us suppose that every point has a weight w_j. Let's define

$$U^{(Y)} = \frac{1}{\sum_{x^{(j)}} w_j} \sum_{i=1}^{p} \sum_{x^{(j)} \in K^{(i)}} w_j \left\|x^{(j)} - y^{(i)}\right\|^2, \tag{2}$$

$$U^{(E)} = \sum_{i=1}^{s} \lambda_i \left\|E^{(i)}(1) - E^{(i)}(0)\right\|^2, \tag{3}$$

$$U^{(R)} = \sum_{i=1}^{r} \mu_i \left\|R^{(i)}(1) + R^{(i)}(2) - 2R^{(i)}(0)\right\|^2. \tag{4}$$

The $U^{(Y)}$ term is the usual average weighted square of distances between $y^{(i)}$ and data points in $K^{(i)}$; $U^{(E)}$ is the analogue of summary energy of elastic stretching and $U^{(R)}$ is the analogue of summary energy of elastic deformation of the net. We can imagine that every node is connected by elastic bonds to the closest data points and simultaneously to the adjacent nodes (see Fig. 3).

Values λ_i and μ_j are coefficient of stretching elasticity of every edge $E^{(i)}$ and coefficient of bending elasticity of every rib $R^{(j)}$. In simple case we have

$$\lambda_1 = \lambda_2 = \ldots = \lambda_s = \lambda(s), \ \mu_1 = \mu_2 = \ldots = \mu_r = \mu(r).$$

To obtain $\lambda(s)$ and $\mu(r)$ dependences we simplify the task and consider the case of evenly stretched and evenly bended net. It is easy to show that if in this case one makes $U^{(R)}$, $U^{(E)}$ independent on the grid "resolution", then

$$\lambda = \lambda_0 s^{\frac{2-d}{d}}, \ \mu = \mu_0 r^{\frac{2-d}{d}} \tag{5}$$

where d is the "internal dimension" of the grid ($d = 1$ in the case of polyline, $d = 2$ in case of hexagonal, rectangular and spherical grids, $d = 3$ in case of cubical grid and so on).

Elastic net approximates the cloud of data points and has regular properties. Minimization of term $U^{(Y)}$ provides approximation, using of $U^{(E)}$ penalizes the total length (or "square", "volume", etc.) of the grid and $U^{(R)}$ is a smoother term, preventing grid from folding and "twisting".

To perform the vertexes optimization step we must derive system of algebraic linear equations to be solved. To start let's consider the situation when we already separated our collection of data points in $K^{(i)}$, $i = 1\ldots p$.

Let's denote

$$\Delta(x, y) = \begin{cases} 1, x = y \\ 0, x \ne y, \end{cases}$$

$$\Delta E^{ij} \equiv \Delta(E^{(i)}(1), y^{(j)}) - \Delta(E^{(i)}(2), y^{(j)}),$$

$$\Delta R^{ij} \equiv \Delta(R^{(i)}(3), y^{(j)}) + \Delta(R^{(i)}(2), y^{(j)}) - 2\Delta(R^{(i)}(1), y^{(j)}).$$

Then differentiation gives

$$\frac{1}{2}\frac{\partial U^{(Y)}}{\partial y^{(j)}} = \frac{1}{\sum\limits_{x^{(i)}} w_i}\left(n_j y^{(j)} - \sum_{x^{(i)} \in K^{(j)}} w_i x^{(i)} \right)$$

$$\frac{1}{2}\frac{\partial U^{(E)}}{\partial y^{(j)}} = \sum_{k=1}^{p} y^{(k)} \sum_{i=1}^{s} \lambda_i \Delta E^{ij} \Delta E^{ik} = \sum_{k=1}^{p} y^{(k)} e_{jk},$$

$$\frac{1}{2}\frac{\partial U^{(R)}}{\partial y^{(j)}} = \sum_{k=1}^{p} y^{(k)} \sum_{i=1}^{r} \mu_i \Delta R^{ij} \Delta R^{ik} = \sum_{k=1}^{p} y^{(k)} r_{jk},$$

where $n_j = \sum\limits_{x^{(i)} \in K^{(j)}} w_i$, $e_{jk} = \sum\limits_{i=1}^{s} \lambda_i \Delta E^{ij} \Delta E^{ik}$, $r_{jk} = \sum\limits_{i=1}^{r} \mu_i \Delta R^{ij} \Delta R^{ik}$. As a result we obtain

$$\frac{1}{2}\frac{\partial D}{\partial y^{(j)}} = \sum_{k=1}^{p} y^{(k)} \left(\frac{n_j \delta_{jk}}{\sum\limits_{x^{(i)}} w_i} + e_{jk} + r_{jk} \right) - \frac{1}{\sum\limits_{x^{(i)}} w_i} \sum_{x^{(i)} \in K_j} w_i x^{(i)} = 0,\ j = 1 \ldots p,$$

and the system of *p linear* equations to find new positions of nodes in multidimensional space $\{y^i, i=1...p\}$:

$$\sum_{k=1}^{p} a_{jk} y^{(k)} = \frac{1}{\sum\limits_{x^{(i)}} w_i} \sum_{x^{(i)} \in K_j} w_i x^{(i)}, \text{ where}$$

$$a_{jk} = \frac{n_j \delta_{jk}}{\sum\limits_{x^{(i)}} w_i} + e_{jk} + r_{jk},\ j = 1 \ldots p, \tag{6}$$

$$\delta_{jk} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}.$$

The values of e_{jk} and r_{jk} depend only on the structure of the grid. If the structure does not change then they are constant. Thus only the diagonal elements of the matrix (6) depend on data set. We must underline that the *a* matrix has sparse structure for a typical grid used in practice. In the Appendix we define this structure, giving an algorithm for calculating only non-zero elements of the matrix.

To minimize the energy of graph *U* we use usual expectation-minimization:

1. Initialize the grid of nodes in data space.
2. Given nodes placement, separate collection of data points to subcollections $K^{(i)}$, $i = 1 \ldots p$.
3. Given this separation, minimize graph energy *U* and calculate new positions of nodes.
4. Go to step 2.

It is evident that this algorithm converges to a final placement of nodes of the grid (energy *U* is a non-decreasing value, and the number of divisions of data points into $K^{(i)}$ is finite). Moreover, theoretically the number of iterations of the algorithm before converging is finite. In practice this number may be too large; therefore we interrupt the process of minimization if change of *U* becomes less than a small value ε or after a fixed number of iterations.

Optimization strategies

As usual, the algorithm provided in the end of the previous section leads only to the local minimum of the functional. Obtaining a solution close to the global minimum can be non-trivial, especially in case when the initial position of the grid is very different from the expected (or unknown) optimal solution. In many practical situations the "annealing" strategy can be used to robustly obtain solutions with low energy levels. In our case this strategy is using "rigid" grids (small length, small bending) with big λ, μ coefficients at the beginning of the learning process and finishing with small λ, μ values (see Fig. 4). Thus, the training goes in several epochs, each epoch with its own grid rigidness. The process of "annealing" promises that the resulting grid will realize the global minimum of energy U or rather close configuration.

Nevertheless, for some artificial distributions (like standard spiral point distribution, used as a test in many papers on principal curves construction) "annealing", starting from any linear configuration of nodes does not lead to the expected solution. In this case, adaptive strategies, like "growing curve" (analogue of what was used by Kegl in his polygonal algorithm [Kegl et al., 1999]) or "growing surface" can be used to obtain suitable configuration of nodes. This configuration does not have to be optimal, in the adaptation process one can still use the grids more rigid than it is needed for good approximation (thus, providing more robust way to do this), finishing the optimization at the next stage with a softer grid (see **spiral** example in the examples section).

Adaptive strategies

The method described above allows to construct different adaptive strategies easily by playing with

- individual λ_i and μ_j weights;
- grid connection topology;
- number of nodes

This is a way to extend the approach significantly making it suitable for practical applications. The ***elmap*** package with implementation of the method described above supports several adaptive strategies that will be described in this section.

First of all let us define a basic operation on the grid, which allows inserting new nodes. Let us denote by **N**, **S**, **R** the sets of all nodes, edges and ribs respectively. Let us denote by **C**(i) set of all nodes which are connected to the ith node by edge. If one has to insert a new node in the middle of an edge I, connecting two nodes k and l, then the following operations have to be accomplished:

1) Delete from **R** those ribs which contain node k or node l ;
2) Delete the edge I from **S**;
3) Put a new node m in **N**;
4) Put in **S** two new edges connecting k and m, m and l;
5) Put in **R** new ribs, connecting m, k and all $i \in \mathbf{C}(k)$, and m, l and all $i \in \mathbf{C}(l)$.

On the step 4, 5 one has to assign new weights to the edges and ribs. This choice depend on the task to be solved. If one constructs "growing" grid, then these weights must be chosen the same as they were at the deleted ones. If one constructs refinement of already constructed grid, one must choose these weights to be twice bigger than they were at the deleted ones.

The *grow-type strategy* is applicable mainly to the grids with planar topology (linear, rectangular, cubic grids). It consists in iterative determining those grid part, which have the largest "load" and doubling the number of nodes in this part of the grid. The load can be defined in different ways.

One natural way is to calculate number of points that are projected in the nodes. For linear grids the grow-type strategy consists in

1) Initializing grid; it must contain at least two nodes and one edge;

2) Determining the edge which has the largest load, by summing the number of data points (or the sum of their weights) projected to the both ends of every edge;

3) Inserting a new node in the middle of the edge, following the operations described above;

4) Optimizing positions of nodes.

One stops this process usually when a certain number of nodes in the grid is reached (see, for example, Kegl et al., 2000), which is connected with the total number of points. In the ***elmap*** package this is an explicit parameter of the method, allowing user to implement his own stopping criterium. Because of this stopping condition the computational complexity is not proportional to the number of objects and, for example, grows like $n^{5/3}$ in the case of Polygonal Line algorithm. Another form of stopping condition is when the MSE does not change more than for a small number ε after several insertion/optimization operations.

The *break*-type adaptive strategy changes individual ribs weights in order to adopt the grid to those regions of data space where the "curvature" of data distribution has a break or is very different from the average. It is particular useful in contour extraction applications of principal curves (see fig. 7). For this purpose the following steps are performed:

1) Collect statistics for the distances from every node *i* to the mean point of the datapoints that are projected into this node: $r_j = \left\| y_j - \frac{1}{\sum_{x^{(i)}} w_i} \sum_{x^{(i)} \in K_j} w_i x^{(i)} \right\|$.

2) Calculate mean and standard deviation for some power of $\boldsymbol{r}$: $m = \overline{r^{\alpha}}$, $s = \sigma_{r^{\alpha}}$; $\alpha>1$ is a parameter and in our experiments is chosen to be 4.

3) Determine those nodes for which $r_j > m+\beta s$, where $\beta>0$ is another parameter, equals to 2 in our experiments.

4) For every node *k* determined at the previous step one finds those ribs that have *k* as their central point and change their weight for $\mu_j^{(new)} = \mu_j^{(old)} \cdot \frac{m}{r_j^{\alpha}}$.

5) Optimize the nodes positions.

6) Repeat this process certain number of times.

Principal graph strategy, implemented in the ***elmap*** package allows performing clustering of curvilinear data features along principal curves. Two example applications of this approach are satellite image analysis [Banfield and Raftery, 1992] or hand-written symbols skeletonization [Kegl and Krzyzak, 2002] (see also fig. 8,9). First, notice that the grid we constructed does not have to be a connected graph. The system matrix (6) is not singular if for every connected component of the graph there are data points that are projected onto one of its nodes. This allows using the same algorithmic kernel to optimize nodes positions of unconnected graph. Notice also that if the sets of edges and ribs are empty, then this algorithm acts exactly like standard K-means clustering.

To construct a contour skeleton, we apply a variant of local linear principal components analysis first, then connect local components into several connected parts and optimize the nodes positions after. This procedure shows to be robust and efficient in applications to clustering along curvilinear features and it was implemented as a part of ***elmap*** package. The following steps are performed:

1) Make a "grid" from a number of unconnected nodes (sets of edges and ribs are empty at this stage). Optimize the nodes positions (i.e., do K-means clustering). The number of nodes is chosen to be a certain proportion of the number of data points. In our experiments we used 5% of the total number of data points. At every iteration of K-means algorithm, those nodes which are "empty" (there is no any datapoint for which the node is the closest one) change their position randomly. After certain number of K-means iterations, empty nodes (or nodes with only one datapoint as well) are removed from the set of all nodes.

2) For every node of the grid in position y^i, local first principal direction is calculated. By local we mean that the principal direction is calculated inside the cluster of datapoints corresponding to the node i. Then this node is substituted by two new nodes in positions $y^{(new1)} = y^i + \alpha s\boldsymbol{n}$, $y^{(new2)} = y^i - \alpha s\boldsymbol{n}$, where $\boldsymbol{n}$ is unit vector in the principal direction, s is the standard deviation of data points belonging to the node i, α is a parameter, which can be taken to be 1. These two nodes are connected by an edge (see fig. 9b).

3) A collection of edges and ribs is generated, following this simple rule: every node is connected to a node which is the closest to this node but not already connected at the step 2, and every such connection generates two ribs, consisting of a new edge and one of the edges made at the step 2.

4) Weights of the ribs are calculated. A rib is assigned a weight equal to $|\cos(\alpha)|$, where α is an intersection angle of two edges this rib contains, if $\alpha \geq 90°$. Otherwise it is zero (or, equally, the rib is eliminated).

5) The nodes positions are optimized.

One possible way to improve the resulting graph further is to apply graph simplification rules, analogously to how it was done in [Kegl et. al, 2002].

The adaptive strategies: "grow", "break" and the principal graphs can be combined and applied one after the other. For example, principal graph strategy can be followed by break-type weight adaptation or by grow-type grid adaptation.

Projecting

In the process of the grid construction we use projection of data into the closest node. This allows to gain in the speed at the data projection step without loosing too much if the grid resolution is good enough. Nevertheless, for presentation of datapoints or for data compression another projectors can be applied. The natural way to do it is to introduce a set of simplexes on the grid (line segments for one-dimensional grids, triangles for two-dimensional, tetrahedron for the 3D grids). Then one performs orthogonal projection onto this set. In order to not calculate all distances to the all simplexes, one can apply a simplified version of the projector: find the closest node of the grid and then consider only those simplexes that contain this node. This type of projection is used in the ***elmap*** package and demonstrated by the example on fig.10.

Since the grid has penalty on it's length (square, volume), the result of the optimization procedure is a bounded manifold, embedded in the cloud of data points. Because of this, if the penalty coefficient is big, many points can have projection on the boundary of the manifold. This can be undesirable, for example, in data visualization applications. To avoid this effect, we introduced in the ***elmap*** package possibility to make a linear extrapolation of the bounded rectangular manifold (extending it by continuity in different directions). Other, more complicated extrapolations can be performed as well, like using Carleman's formulas (see [Gorban et al, 2002;Aizenberg, 1993]).

Examples

On the figure 5 we present two examples of 2D-datasets provided by Kegl at http://www.iro.umontreal.ca/~kegl/research/pcurves/implementations/Samples/.

The first dataset called **spiral** is one of the standard in the principal curves literature ways to show that one's approach has better performance than the initial algorithm provided of Hastie and Stuelze. As we already mentioned, this is a bad case for optimization strategies, which start from linear distribution of nodes and try to optimize all the nodes together in one loop. But the adaptive "growing curve" strategy, though being by order of magnitude slower than the "annealing", finds the solution quite stably, with exception for the region in the neighborhood of zero, where the spiral has very different (comparing to the average) curvature.

Second dataset, called **large** is a simple case, despite the fact that it has comparatively large sample size (10000 points). The nature of this simplicity is in that the initial first principal component based approximation is already effective; the distribution is in fact *quasilinear*, since the principal curve can be unambiguously orthogonally projected onto a line. On fig.5b it is shown that the generating curve, which was used to generate this dataset, has been discovered almost perfectly and in very short time. To give the idea of speed, we mention that in case of the simplest optimization (one epoch with fixed grid rigidness, which is suitable in case of a good initial approximation) the algorithm we described gives the principal curve, approximated by 100 nodes in less than 0.5 seconds on a computer with Athlon 1800 MHz processor. Application of annealing strategy with 4 epochs gives principal curve approximately in 1.5 seconds on the same computer.

Third example illustrates modeling of surfaces in 3D. The interesting challenge is to model **molecular surfaces** of complex biological molecules like proteins using principal manifold approach. We extracted Van-der-Waals molecular surface, using slightly modified Rasmol [Sayle and Bissell, 1992] source code (available from authors by request) for a simple fragment of DNA. The topology of the surface is expected to be spherical. We must notice that since it is impossible to make the lengths of all edges equal for the sphere-like grid, in the ***elmap*** package some corrections are performed for edge and rib weights during the grid initialization (shorter edges are given with larger weights proportionally and the same for the ribs). As a result one gets smooth principal manifold with a spherical topology approximating rather complicated set of points. This allows also to introduce a global spherical coordinate system on the molecular surface. The advantage of the method is its ability to deal with not only star-like shapes as spherical harmonic functions approach does (see, for example, [Cai et al., 2002]) but also to model complex forms with cavities as well as non-spherical forms. The result of applying principal manifolds construction by elmap package is shown on fig.6.

The forth example is to demonstrate extracting of curvilinear features from images with **elmap** package. Figures 8 and 9 demonstrate how "principal graph" strategy is used for clustering datapoints along curvilinear features or for skeletonization of hand-written symbols.

Our final, fifth example, illustrates application of the principal manifold method in multidimensional **data visualization** and dimension reduction. As in the case of molecular surface modeling, we take an example of a dataset from bioinformatics. The genome of C.eleganse (small worm with only one-hundred cells) contains approximately 17000 genes, each of them can be characterized by it's codon usage (there are 64 codons, i.e. triplets of 4 genetic letters, this gives 64-dimensional vector of their frequencies), dinucleotide and nucleotide usage (this gives additional 20 dimensions). The resulting dataset has 17083 points with 84 dimensions. PCA view of the dataset is shown at fig.10a. To make noise-filtering, the dataset was projected first into 25-dimensional space spanned by the first 25 principal vectors. In this space, using our ***elmap*** package, we constructed a two dimensional principal surface, approximated by 1296 nodes. The datapoints were projected onto the manifold using projecting in the closest point of the manifold (as proposed above). Using 3-epoch optimization strategy, provided in the sample initialization file for the ***elmap*** package, it

takes 300 seconds to do this on a computer with Athlon 1800 MHz processor. The initial MSE, obtained by a principal plane approximation was 4.59. The resulting manifold provides MSE about 3.60; what is at 22% better than approximation by the principal plane (this value is relatively big, having in mind that we approximate 25-dimensional dataset). The resulting projection image is shown on the fig. 10b. Changing point forms/sizes we marked two signals that are clearly seen on this plot. More detailed analysis shows that indeed these two groups of points (genes) have very special position in dataspace (i.e., codons and dinucleotide compositions) with respect to the main cluster of data. The principal manifold we constructed can be utilized for displaying different functions defined in the dataspace. On fig.10c visualization of a simple non-parametric estimation of the density distribution is shown. One can see that in general the non-linear manifold captures more essential features of the dataset than the PCA plot.

Method implementation

In the implementation of the algorithm we used SparseLib [Dongarra et al., 1994] library together with IML++ library to store the matrix and to solve the system of linear equations. We used BLAS kernel provided by the authors of SparseLib without any platform-specific optimization. This combination showed rather good performance characteristics, still being easily portable, i.e. written, using ANSI standards.

Discussion

We introduced a new algorithmic kernel for calculating grid approximations for principal manifolds of different topologies and dimensions. The main advantages of the method are speed and good performance. The optimization criterium we formulated has particular simple form and natural physical interpretation. Together with usual mean square node-to-point distance term our minimized functional contains two penalizing terms: $U^{(E)}$ and $U^{(R)}$, both quadratic with respect to the grid nodes positions. As one can see from (3) and (4) they are similar to the sum of squared grid approximations of the first and second derivatives, in the directions, guided by natural choice of ribs [1]. The $U^{(E)}$ term penalizes the total length (or square, volume) of the principal manifold and, indirectly, makes the grid regular by penalizing non-equidistant distribution of nodes along the grid. The $U^{(R)}$ term is a smoothening factor, chosen to be quadratic instead of using cosinus function as in the algorithm of Kegl [Kegl et. al, 1999].

Good characteristics of the method such as its universality, low computational complexity and inherited parallelism open new fields to the applications of principal manifolds, especially for the analysis of huge datasets with hundreds of thousands of points with dimensionality of order of hundreds. The algorithm we described with its C++ implementation provide a way to construct a principal manifolds for these datasets approximated by number of nodes of order of 10000 in a reasonable time.

Appendix. Constructing the sparse matrix

The matrix (6) has p^2 number of elements (p – number of grid nodes), but for typical grids only kp of them are non-zero, where $k << p$. Here we provide a simple procedure to fill only non-zero elements of the matrix, thus, define its sparse structure.

For the e^{jk} matrix:

[1] Of course, the differences must be divided by node-to-node distances in order to be true derivative approximations, but in this case the quadratic structure of the term would be violated. We suppose that the grid is regular with almost equal node-to-node distances, then the dependence of coefficients λ_i, μ_j on the total number of nodes contains this factor.

1) all e^{jk} values are initialized by zero;
2) if for an edge E^i with weight λ_i, the beginning node is $y^{k1}$and the ending node is y^{k2}, then we update the e^{jk} values:

$$e^{k_1k_1} = e^{k_1k_1} + \lambda_i,\ e^{k_2k_2} = e^{k_2k_2} + \lambda_i,$$

$$e^{k_1k_2} = e^{k_1k_2} - \lambda_i,\ e^{k_2k_1} = e^{k_2k_1} - \lambda_i.$$

3) Steps 1-2 are repeated for every edge.

For the r^{jk} matrix:

1) all r^{jk} values are initialized by zeros;
2) if for an edge R^i with weight μ_i, the beginning node is y^{k1}, the middle node is y^{k2} and the ending node is y^{k3}, then we update the r^{jk} values:

$$r^{k_1k_1} = r^{k_1k_1} + \mu_i,\ r^{k_2k_2} = r^{k_2k_2} + 4\mu_i,\ r^{k_3k_3} = r^{k_3k_3} + \mu_i$$

$$r^{k_1k_2} = r^{k_1k_2} - 2\mu_i,\ r^{k_2k_1} = r^{k_2k_1} - 2\mu_i,$$

$$r^{k_2k_3} = r^{k_2k_3} - 2\mu_i,\ r^{k_3k_2} = r^{k_3k_2} - 2\mu_i,$$

$$r^{k_1k_3} = r^{k_1k_3} + \mu_i,\ r^{k_3k_1} = r^{k_3k_1} + \mu_i.$$

3) Steps 1-2 are repeated for every edge.

References

Aizenberg, L. Carleman's Formulas in Complex Analysis: Theory and Applications. Mathematics and Its Applications; **244**. Kluwer, 1993.

Banfield J. D., Raftery A.E. Ice floe identification in satellite images using mathematical morphology and clustering about principal curves. *Journal of the American Statistical Association* **87**, N 417, pp. 7-16, 1992.

Cai W., Shao X., Maigret B. Protein-ligand recognition using spherical harmonic molecular surfaces: towards a fast and efficient filter for large virtual throughput screening. J. Mol. Graph. Model. 2002 Jan; 20(4): 313-28.

Dongarra J., Lumsdaine A., Pozo R., Remington K. A Sparse Matrix Library in C++ for High Performance Architectures. Proceedings of the Second Object Oriented Numerics Conference, pp. 214-218, 1994.

Erwin E., Obermayer K., Schulten K. Self-organizing maps: ordering, convergence properties and energy functions. *Biological Cybernetic*s, 67:47–55, 1992.

Gorban A.N., Pitenko A.A., Zinov'ev A.Y., Wunsch D.C. Vizualization of any data using elastic map method. 2001. *Smart Engineering System Design* **11**, p. 363-368.

Gorban A.N., Rossiev A., Makarenko N., Kuandykov Y., Dergachev V. Recovering data gaps through neural network methods. International Journal of Geomagnetism and Aeronomy, **2002**, V 3, N. 2.

Gorban A.N., Zinovyev A. Yu. Method of Elastic Maps and its Applications in Data Visualization and Data Modeling. *International Journal of Computing Anticipatory Systems*, CHAOS. 2001. V **12**. PP. 353-369.

Gorban A.N., Zinovyev A., Wunsch D.C. Application of the method of elastic maps in analysis of genetic texts. 2003, *in press*.

Gorban A.N., Zinovyev A.Yu. Visualization of data by method of elastic maps and its applications in genomics, economics and sociology. Preprint of Institut des Hautes Etudes Scientiques. 2001. M/01/36. http://www.ihes.fr/PREPRINTS/M01/Resu/resu-M01-36.html

Hastie T. Principal curves and surfaces. *PhD Thesis.* Stanford University, 1984.

Hastie T., Stuetzle W. Principal curves. *Journal of the American Statistical Association* **84**, N 406. pp. 502-516, 1989.

Kegl B. Principal curves: learning, design, and applications, *Ph. D. Thesis,* Concordia University, Canada, 1999.

Kegl B., Krzyzak A. Piecewise linear skeletonization using principal curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**, N 1, pp. 59-74, 2002.

Kegl B., Krzyzak A., Linder T., Zeger K. A polygonal line algorithm for constructing principal curves. *Neural Information Processing Systems 1998.* MIT Press, 1999, pp. 501-507.

Kegl B., Krzyzak A., Linder T., Zeger K. Learning and design of principal curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**, N 3, pp. 281-297, 2000.

LeBlanc M., Tibshirani R. Adaptive principal surfaces. *Journal of the American Statistical Association* **89**, pp. 53-64, 1994.

Mulier F., Cherkassky V. Self-organization as an iterative kernel smoothing process. *Neural Computatio*n, 7:1165–1177, 1995.

Ritter H., Martinetz T., Schulten K.. *Neural Computation and Self-Organizing Maps: An Introductio*n. Addison-Wesley, Reading, Massachusetts, 1992.

Sayle R., Bissell A. RasMol: A Program for Fast Realistic Rendering of Molecular Structures with Shadows. 1992. In *Proceedings of the 10th Eurographics UK'92 Conference*, *University of Edinburgh, Scotland.*

Stanford D., Raftery A.E. Principal curve clustering with noise. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**, N 6, pp.601-609, 2000.

Verbeek J.J., Vlassis N., Krose B. A k-segments algorithm for finding principal curves**.** A technical report**.** http://citeseer.nj.nec.com/article/verbeek00ksegments.html.

Figure 1. Node, edge and rib

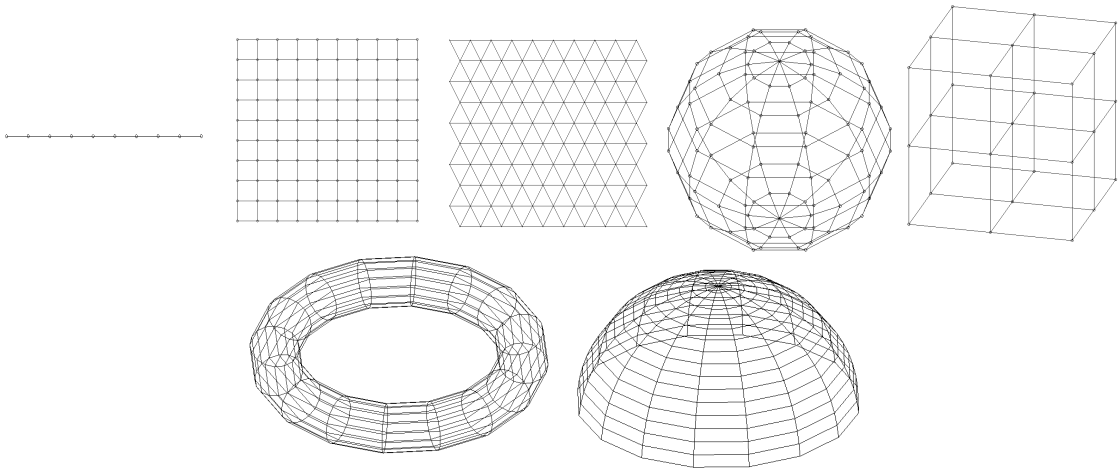


Figure 2. Elastic nets used in practice

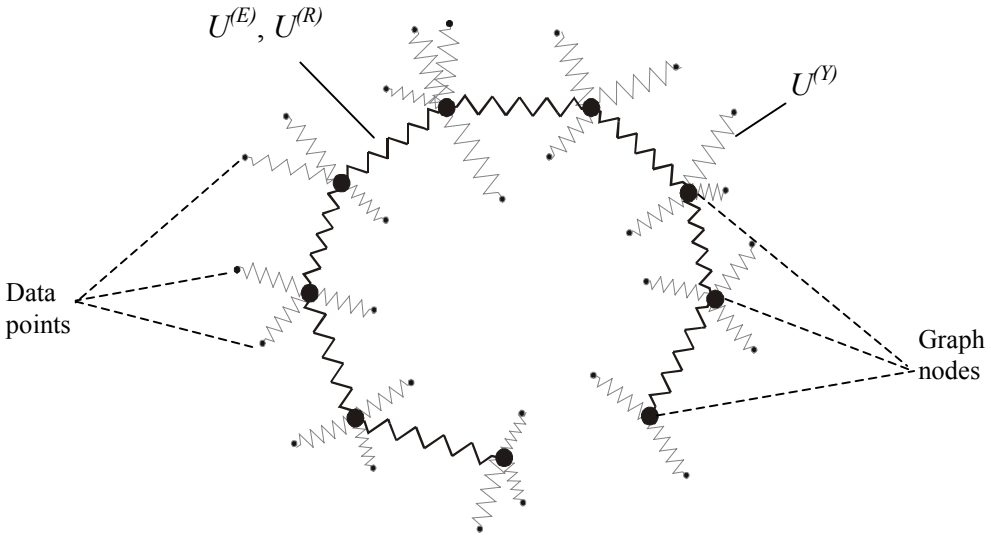


Figure 3. Energy of elastic net

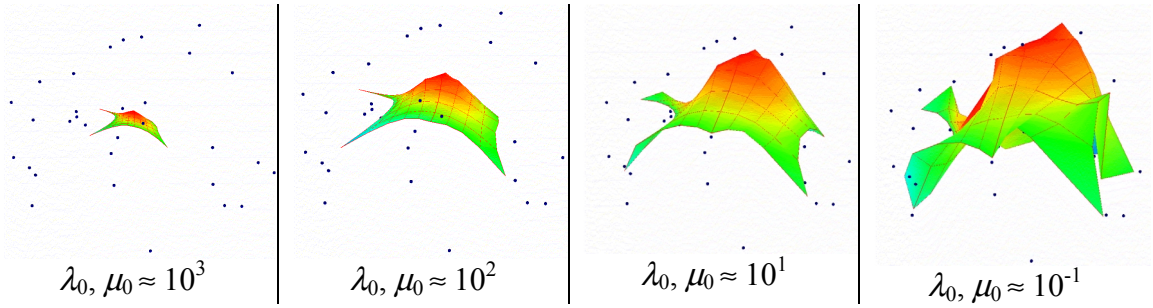
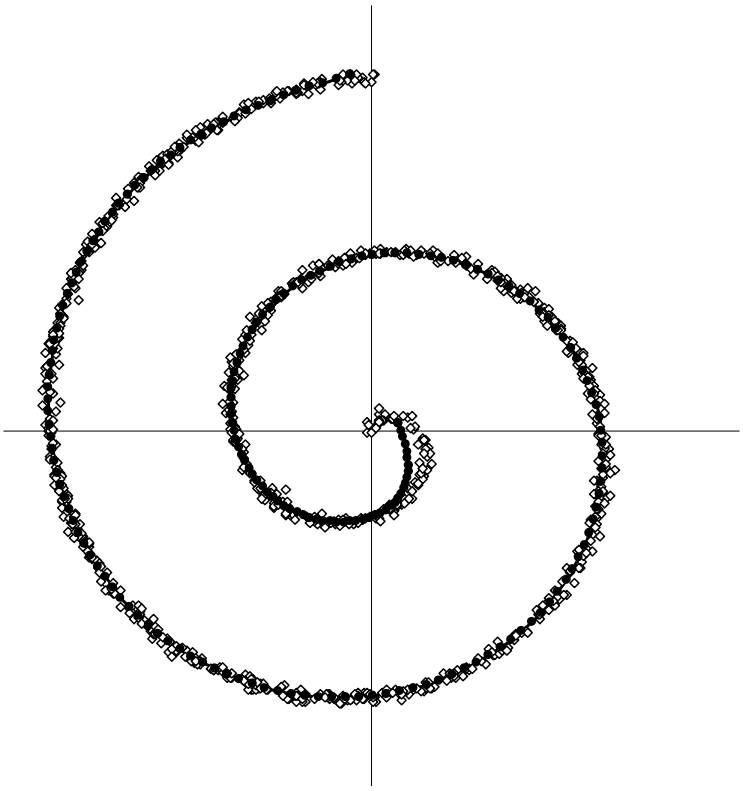
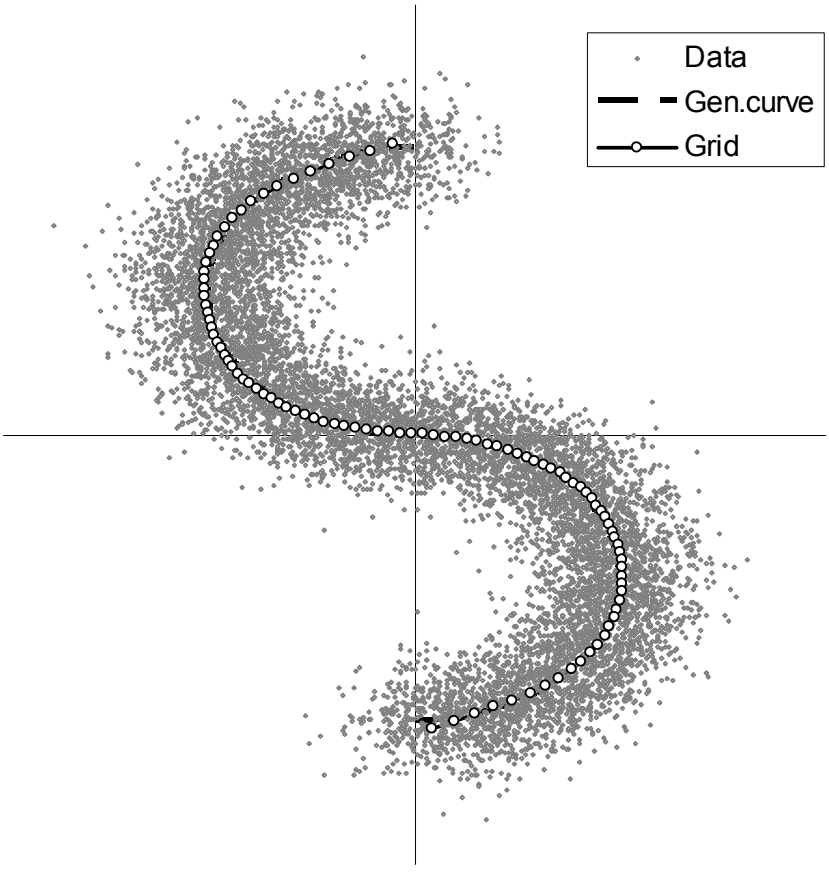


Figure 4, Training elastic net in several epochs



a) **spiral**



b**) large**

Figure 5. Two-dimensional examples of principal curves construction

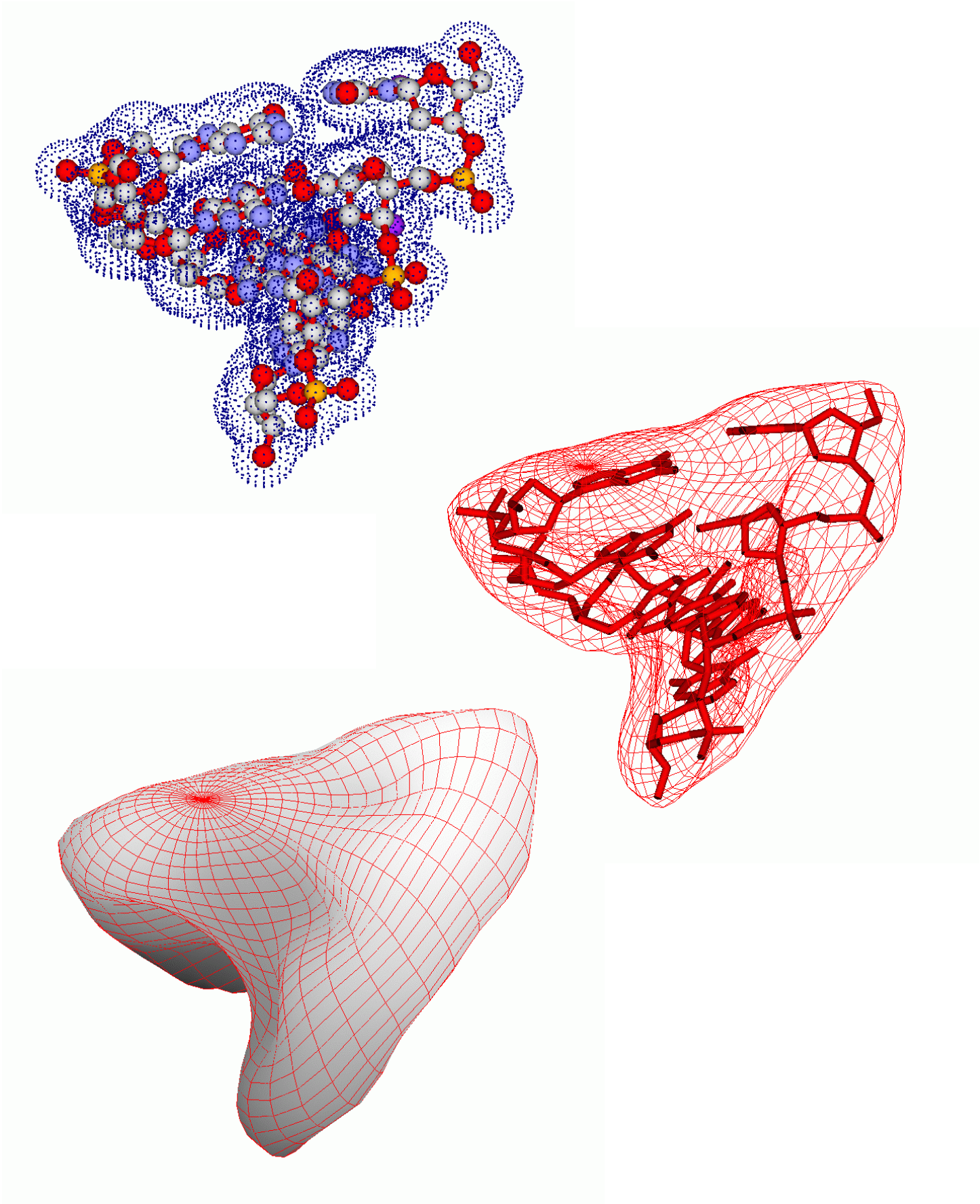


Figure 6. Construction of principal surface with spherical topology for a distribution of points on Van der Waals molecular surface of a biological molecule.

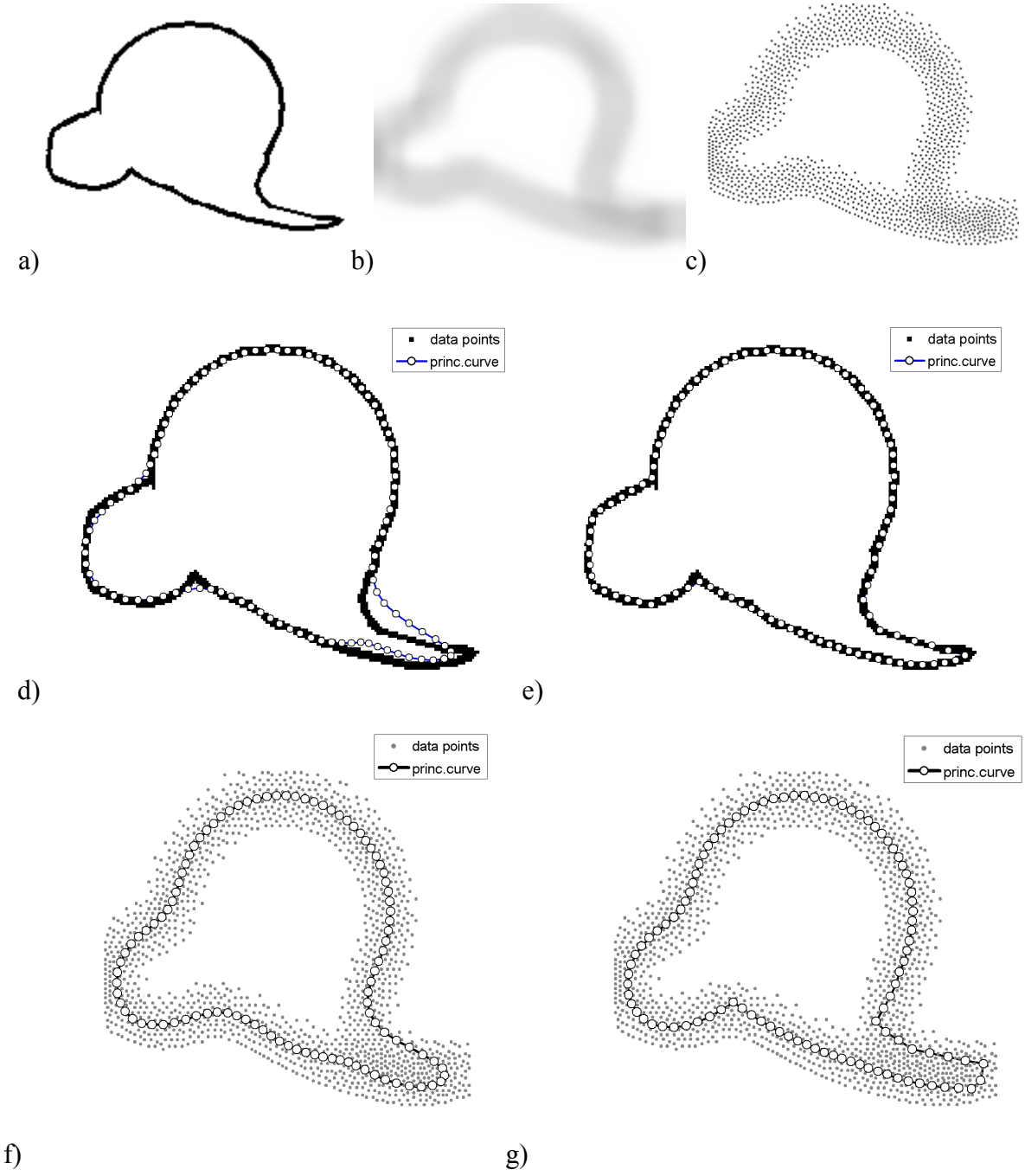


Figure 7. Contours extraction with closed principal curve.
a) initial countour; b) blurred contour; c) Floyd-Steinberg error diffusion color image binarization; d,f) fitting closed principal curve with constant "elasticity", regions of higher curvature can not be fitted equally well; e,g) fitting closed principal curve with adaptive elasticity ("break" adaptation dtrategy).

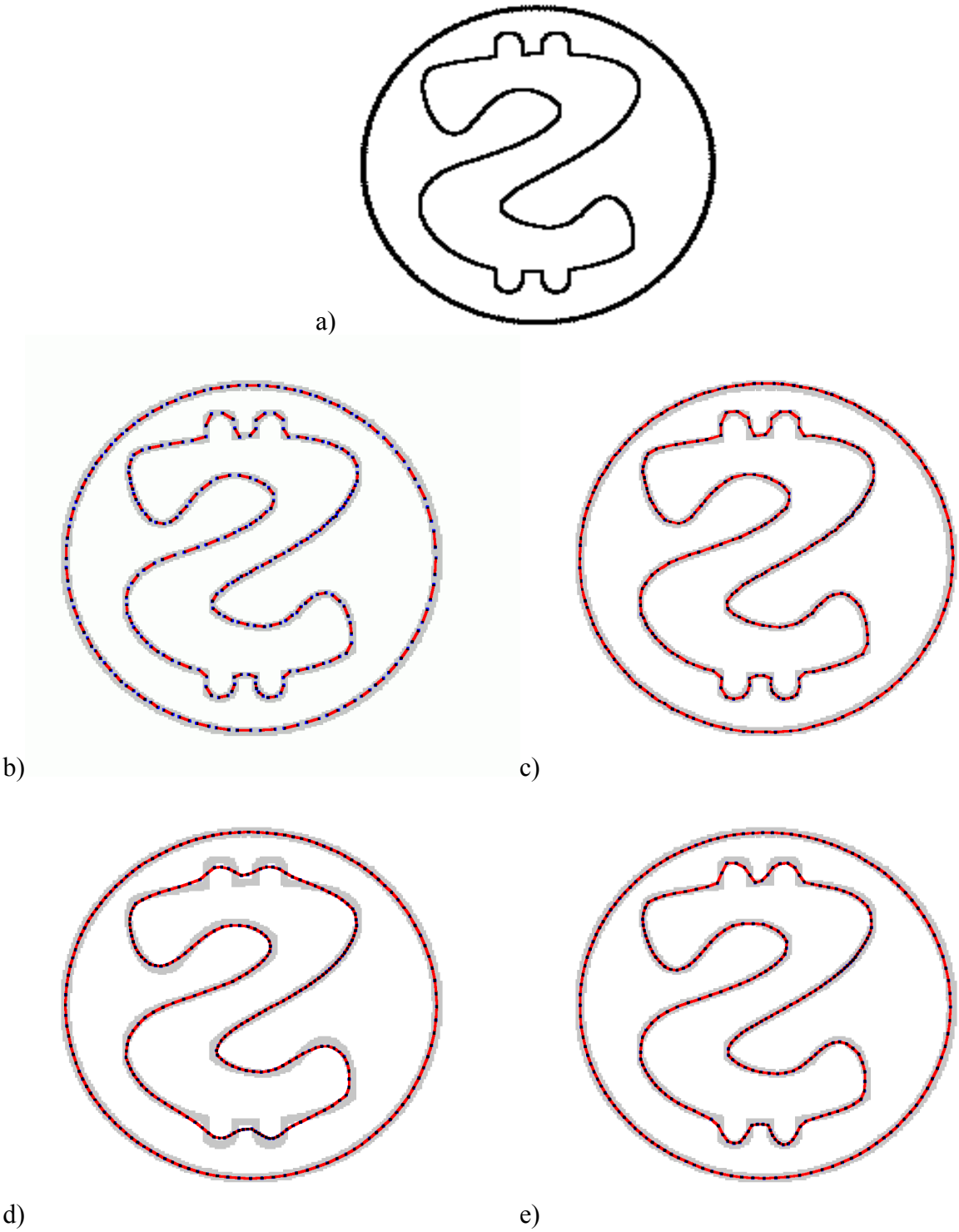


Figure 8. Non-linear features extraction using principal curves
a) initial image;
b) calculation of local principal components;
c) connecting the graph;
d,e) graph vertices optimization with principal components algorithm.

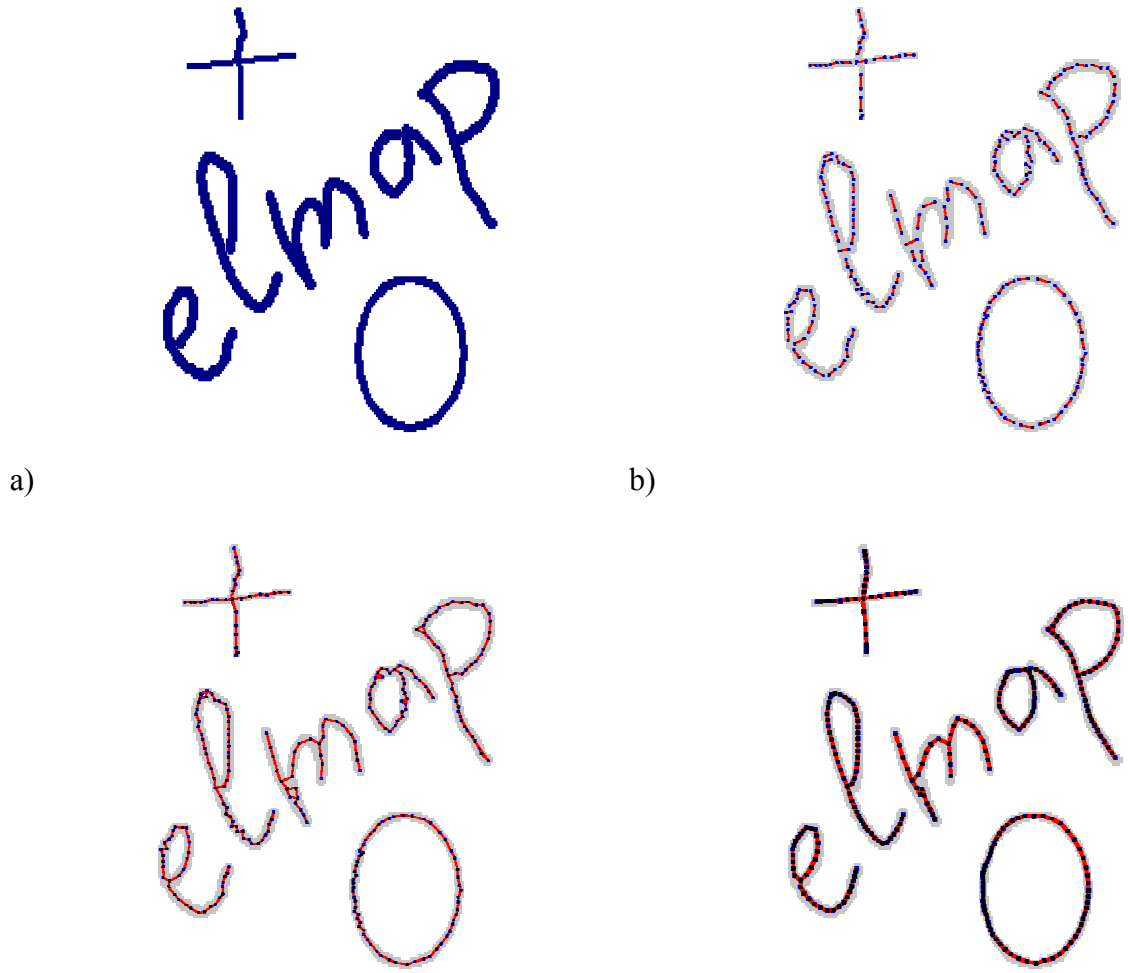


Figure 9. Skeletonization using principal curves.
a) initial image;
b) calculation of local principal components;
c) connecting the graph;
d) graph vertices optimization with principal components algorithm.

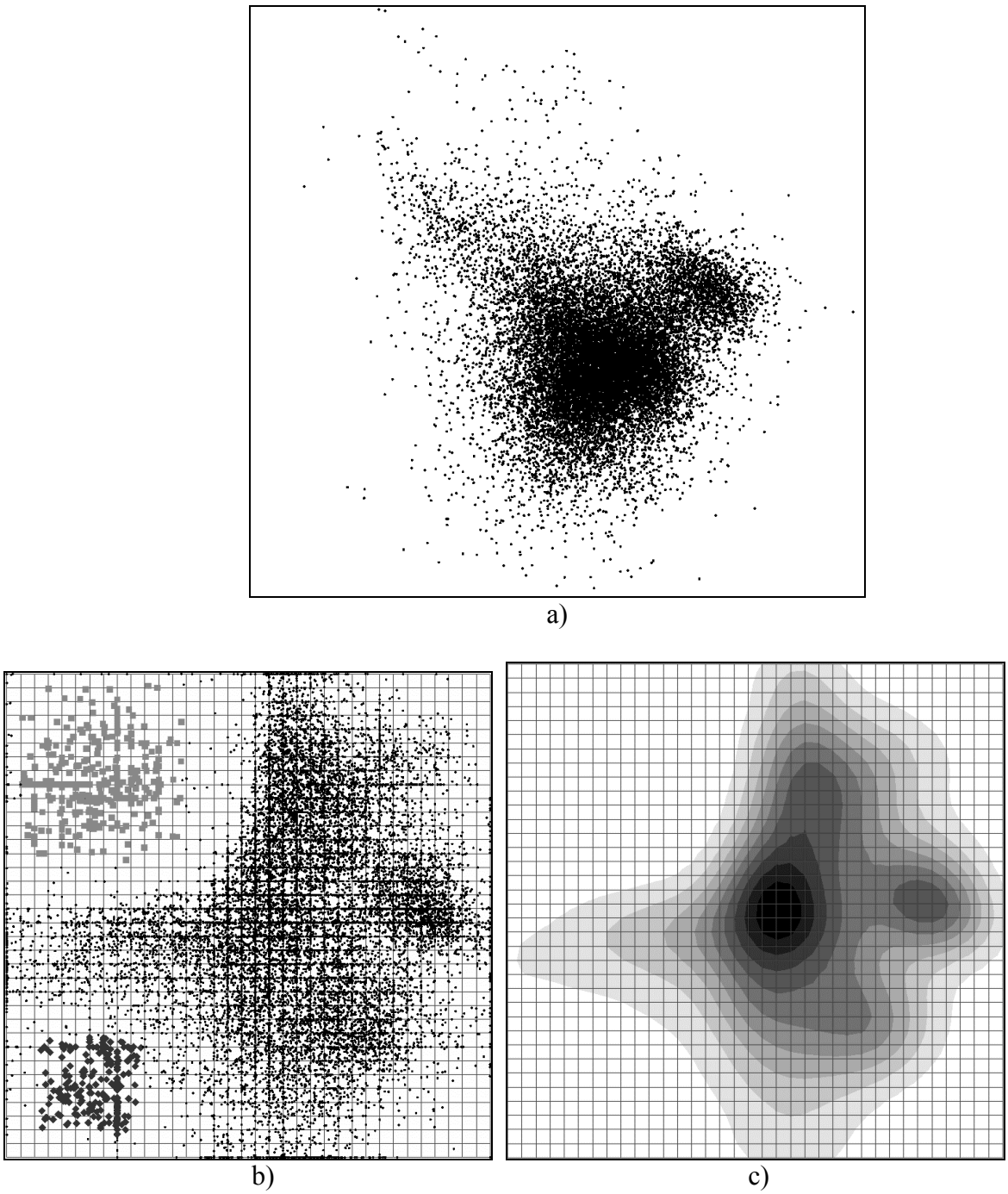


Figure 10. Visualization of a big dataset in 84-dimensional space.
a) PCA view;
b) projection onto the manifold constructed;
two strong signals are marked by changing point sizes/forms;
c) principal manifold as a screen for displaying points density distribution.